
Table of Contents

Introduction	1.1
What is Machine Learning?	1.2
Types of Machine Learning	1.3
Supervised Learning	1.3.1
Notations	1.3.1.1
Probabilistic Modeling	1.3.1.2
Naive Bayes Classifier	1.3.1.2.1
Linear Regression	1.3.1.3
Nearest Neighbor	1.3.1.4
Evaluating a Classifier	1.3.1.5
Parametric Estimation	1.3.1.6
Bayesian Approach to Parameter Estimation	1.3.1.6.1
Parametric Estimation for Simple Linear Regression	1.3.1.6.2
Parametric Estimation for Multivariate Linear Regression	1.3.1.6.3
Parametric Estimation for Simple Polynomial Regression	1.3.1.6.4
Parametric Estimation for Multivariate Polynomial Regression	1.3.1.6.5
Bias and Variance of an Estimator	1.3.1.7
Bias and Variance of a Regression Algorithm	1.3.1.8
Model Selection	1.3.1.8.1
Logistic Regression	1.3.1.9
Decision Trees	1.3.1.10
Using Decision Trees for Regression	1.3.1.10.1
Bias and Variance	1.3.1.10.2
Dimensionality Reduction	1.3.1.11
Neural Networks	1.3.1.12
Training a Neuron	1.3.1.12.1
MLP	1.3.1.12.2
Regression with Multiple Outputs	1.3.1.12.2.1
Advice/Tricks and Issues to Train a Neural Network	1.3.1.12.2.2
Deep Learning	1.3.1.12.3

Support Vector Machines	1.3.1.13
Ensemble Learning	1.3.1.14
Unsupervised Learning	1.3.2
K-Means Clustering	1.3.2.1
Probabilistic Clustering	1.3.2.2
Reinforcement Learning	1.3.3

CS-GY 6923: Machine Learning

This book contains my notes for the "Machine Learning" (CS-GY 6923) course that I had taken at NYU Tandon School of Engineering in the Fall of 2018, towards my MS in Computer Science.

The contents of this book have been acquired from several sources, including the presentation slides provided by Prof. Lisa Hellerstein.

Some prerequisites include an understanding of:

- Probability and Statistics
- Simple Differentiation and Integration (especially for polynomials)
- Partial Differentiation
- Chain Rule
- Linear Algebra
- Eigenvectors and Eigenvalues

What is Machine Learning?

Machine Learning is the field of study that deals with making computers "learn" from data/experience.

Main Use Cases

Machine Learning is mainly useful in cases where:

- there is no human expertise available to solve a problem
- there is human expertise, but humans cannot explain the expertise well enough (if at all) to be able to hard-code a program
- the solution changes with time
- a customized solution is required, ex. targeted marketing/advertising

How is Machine Learning Getting Better?

Advances in ML are mainly owed to:

- the increased amounts of data
- the development of better algorithms
- the availability of cheap and more powerful computing resources

Types of Machine Learning

There are mainly 4 types of Machine Learning:

1. Supervised Learning
2. Unsupervised Learning
3. Reinforcement Learning
4. Learning Association Rules (Data Mining)

Supervised Learning

In this kind of learning, we assume that the training examples are labeled.

There are two kinds of Supervised Learning problems:

- Classification
- Regression

Classification

It refers to the task of assigning a label to a given example i.e. to categorize it into one of multiply categories. Ex. spam/not spam, family car/not family car

Here, the output is discrete.

Regression

It refers to the task of predicting a real-valued output. Ex. prediction house value, temperature

Here, the output is continuous.

Notations

The training set is denoted by X .

It has N training examples.

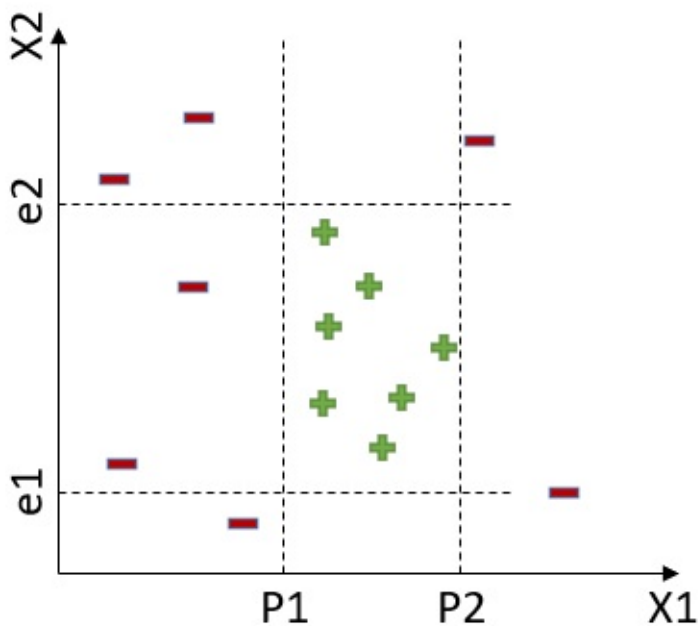
Each example is denoted by x^t, r^t where x^t is the feature set of the t^{th} training example and r^t is the corresponding label.

$$x^t = \begin{bmatrix} x_1^t \\ x_2^t \\ \cdot \\ \cdot \\ \cdot \\ x_d^t \end{bmatrix}$$

The training set is denoted as $X = \{x^t, r^t\}_{t=1}^N$

$h(x)$ is the hypothesis that assigns a label r to x . For example, if we have a task of classifying cars as family cars/not family cars, based on two features X_1, X_2 , based on the below feature space, we could hypothesize that:

$$h(x) = \begin{cases} 1; P_1 \leq X_1 \leq P_2 \text{ \& } e_1 \leq X_2 \leq e_2 \\ 0; otherwise \end{cases}$$

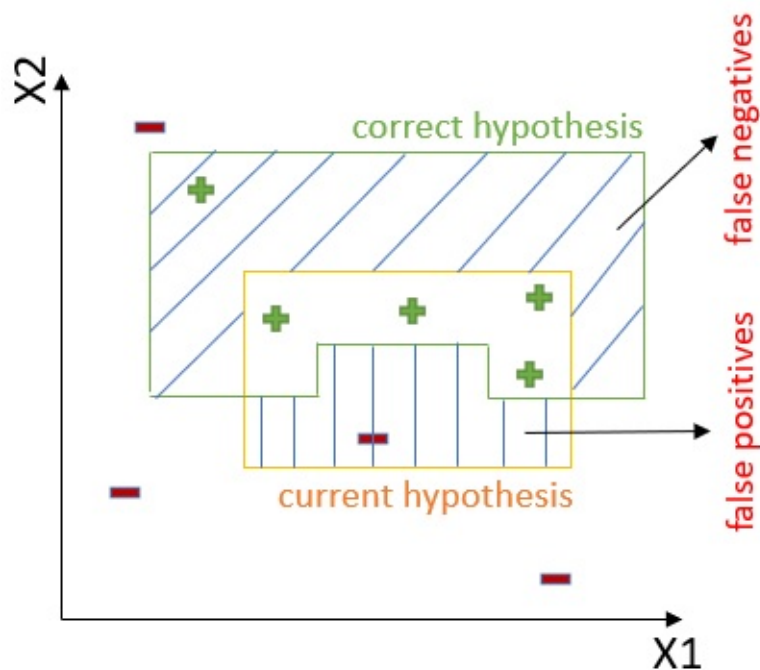


This hypothesis, however, may or may not be correct.

The error of the hypothesis h on X is given by:

$E(h|X)$ or $Err(h|X) = |\{x^t \in X | h(x^t) \neq r^t\}|$, basically the number of misclassified examples.

Say we know the correct hypothesis and we compare our current hypothesis with the correct hypothesis:



The current hypothesis labels everything inside the orange box as + and everything outside as -.

False positives are examples that are mistakenly labeled by our current hypothesis as positive. False negatives are examples that are mistakenly labeled by our current hypothesis as negative.

Based on the task at hand, we must focus on reducing either false positives or false negatives.

Probabilistic Modeling

This refers to models of data generation i.e. **generative models**. They model where the data comes from.

Consider spam classification. We would learn the probability distribution for the examples in the spam class as well as the probability distribution for the examples in the non-spam (ham) class.

Given a new sample x , we must then calculate $P(x \text{ is spam})$.

By default, we label it as spam if $P(x \text{ is spam}) \geq 0.5$ i.e. if $P(x \text{ is spam}) \geq P(x \text{ is ham})$.

More generally put, we must compute $P(C|X)$ i.e. the probability of a class C given a training example X i.e. the probability of X belonging to C .

Bayes' Rule

According to Bayes' Rule,

$$P(C|X) = \frac{P(C)P(X|C)}{P(X)}$$

$P(C)$ is the **prior** probability, $P(X|C)$ is the **likelihood** probability of X being generated from C and $P(X)$ is known as the **evidence**. $P(C|X)$ is called the **posterior** probability.

C is the **hypothesis** and X is the **data**.

The prior probability is computed without having seen the data X .

For example:

X - I am late to class

C_1 - I was kidnapped by Martians

C_2 - I was thinking about research and I lost track of time

Say, $P(C_1) = 0.00000...1$. We assume that these are the only two hypotheses. Therefore, $P(C_2) = 0.999999...9$.

The probability that I was late to class *given* that the Martians kidnapped me is pretty high.

So, say $P(X|C_1) = 0.97$. Now, the probability that I was late to class *given* that I was thinking about research and lost track of time is also pretty high. So, say $P(X|C_2) = 0.5$.

Bayes' Rule aims to find the most probable hypothesis. In other words, given X , we aim to choose the hypothesis that maximizes $P(C|X)$ i.e. $\operatorname{argmax}_C P(C|X)$. This is also called the **Maximum a-posteriori (MAP)** hypothesis, where a-posteriori means 'after' seeing the data.

$\operatorname{argmax}_C P(C|X) = \operatorname{argmax}_C \frac{P(C)P(X|C)}{P(X)} = \operatorname{argmax}_C P(C)P(X|C)$ since the denominator is a constant.

In our example, $P(C_1|X) = P(C_1)P(X|C_1) = 0.0000...1 * 0.97$ and

$P(C_2|X) = P(C_2)P(X|C_2) = 0.999...9 * 0.5$

Therefore, we choose hypothesis C_2 .

The **Maximum Likelihood (ML)** hypothesis is given by $\operatorname{argmax}_C P(X|C)$.

Note that if we have a **uniform prior** (distribution) i.e. all the hypotheses are equally likely, the MAP and ML hypotheses are the same.

Continuous Distributions and Bayes' Rule

For continuous distributions, i.e. for a continuous random variable X , we cannot compute $P(X)$ directly, because X doesn't hold a fixed set of discrete values.

Instead, we compute the pdf(X) i.e. probability distribution function of X . We denote it as $p(X)$. It is, visually speaking, the height of the curve at X .

Say we have two probability distributions, one for mens' heights and the other for womens' heights. They are assumed to be Normal/Gaussian distributions.

Say, $C_1 : \text{man}, C_2 : \text{woman}; P(C_1) = P(C_2) = 0.5$. So, ML hypothesis = MAP hypothesis.

For continuous random variables X , the Bayes' Rule is as follows:

$$P(C|X) = \frac{P(C)p(X|C)}{p(X)}$$

The MAP hypothesis = $\operatorname{argmax}_C P(C|X) = \operatorname{argmax}_C P(C)p(X|C)$

Naive Bayes Classifier

Say our data has d attributes/features, and that these attributes are binary. Also, we have a finite set of classes C .

$$C_{MAP} = \operatorname{argmax}_{c \in C} P(X|C)P(C)$$

The main issue here is that we do not know $P(X|C)$ in advance, and it is difficult, if not impossible, to model the joint distribution of all the d attributes. Therefore, we use the simplifying assumption that the attributes are **conditionally independent** i.e. the attributes are independent, *given the class*. This assumption is also called **class-conditional independence**.

$$\text{So, } P([x_1, x_2, \dots, x_d]|C) = P(x_1|C) * P(x_2|C) * \dots * P(x_d|C)$$

Therefore, we calculate C_{MAP} as:

$$C_{MAP} = \operatorname{argmax}_C P(x_1|C) * P(x_2|C) * \dots * P(x_d|C) * P(C)$$

Since we need to compute a product of probabilities, we must prevent any of the individual probabilities from being 0. This would nullify the effect of all the other probabilities.

To do so, we use **smoothing**.

If N is the total number of examples having a certain class C , and t is the number of times $x_i = v$, then the non-smoothed estimate is given by:

$$P(x_i = v|C) = \frac{t}{N}$$

If $t=0$, this probability becomes 0!

So, we perform **add-m smoothing**:

$$P(x_i = v|C) = \frac{t+m}{N+ms} \text{ (where } s \text{ is the number of possible values for the attribute } x_i \text{).}$$

Sometimes, we use $m=1$. This is called **add-1 smoothing**. In most cases, we limit m to the range $(0,1]$.

Linear Regression

As discussed earlier, Regression is a Supervised Learning technique that is used to predict a **real value**.

Given a dataset, $X = \{X^t, r^t\}_{t=1}^N$, our aim is to find the line that minimizes the **Mean Squared Error**.

The function that we want to learn can be denoted as:

$$f(x|\theta) = w_0 + w_1 X$$

We need to find the best values for w_0, w_1 (denoted using the term θ , meaning parameters) using the available training data.

The Mean Squared Error of f on the data set X is given by:

$$Err(f|X) = \frac{1}{N} \sum_{t=1}^N (r^t - f(x^t))^2$$

Note that the **absolute error** would be: $\frac{1}{N} \sum_{t=1}^N |r^t - f(x^t)|$

One advantage of using MSE instead of absolute error is that it will penalize a line more for being further away from the data points. However, this makes it sensitive to outliers i.e. a line would be penalized for being far away from outliers, although it shouldn't be. Another reason for using MSE over absolute error is that MSE is continuous while absolute error is not. It is easier to minimize a continuous function by taking the derivative.

The Role of Noise

When we obtain a data set, we assume that the values were computed with an instrument that was susceptible to noise. It is generally assumed that the noise follows a Gaussian distribution.

Therefore, r^t will not be exactly equal to $f(x^t)$. Instead:

$$r^t = f(x^t) + \epsilon^t$$

where the noise $\epsilon^t \sim N(0, \sigma^2)$

We assume that each ϵ^t is independent.

Noise also affects the probability:

$$p(r^t|x^t) = N(f(x^t), \sigma^2)$$

Now, we want to find the ML (Maximum Likelihood) value for θ i.e. the ML 'estimator' f.

$$= \operatorname{argmax}_{\theta} p(X|\theta) = \operatorname{argmax}_{\theta} \prod_{t=1}^N p(r^t|x^t, \theta) \text{ --- (1)}$$

Recall that if $X \sim N(\mu, \sigma^2)$, then the pdf for the distribution on X is given by:

$$p(X) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(X-\mu)^2}{2\sigma^2}}$$

If we use this formula in (1), it might make things complicated. Instead, we use the **log trick**.

We can do this because argmax is unaffected by log, for non-negative valued functions.

Using a log turns products into sums and gets rid of exponents, making things less complicated.

Now, (1) becomes:

$$\begin{aligned} \operatorname{argmax}_{\theta} \log \prod_{t=1}^N p(r^t|x^t, \theta) &= \operatorname{argmax}_{\theta} \sum_{t=1}^N \log p(r^t|x^t, \theta) = \operatorname{argmax}_{\theta} \sum_{t=1}^N \log \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(r^t - (w_1 x^t + w_0))^2}{2\sigma^2}} \\ &= \operatorname{argmax}_{\theta} \sum_{t=1}^N \log \frac{1}{\sqrt{2\pi}\sigma} \text{ (can be ignored)} + \sum_{t=1}^N \frac{-(r^t - (w_1 x^t + w_0))^2}{2\sigma^2} = \operatorname{argmin}_{\theta} \sum_{t=1}^N (r^t - (w_1 x^t + w_0))^2 \\ &\text{(ignoring the denominator).} \end{aligned}$$

This proves that, under the assumption of independent additive Gaussian noise, the line that denotes the ML estimator (i.e. the ML hypothesis) is the same line that minimizes the MSE.

Nearest Neighbor

Idea: Similar examples tend to have the same label.

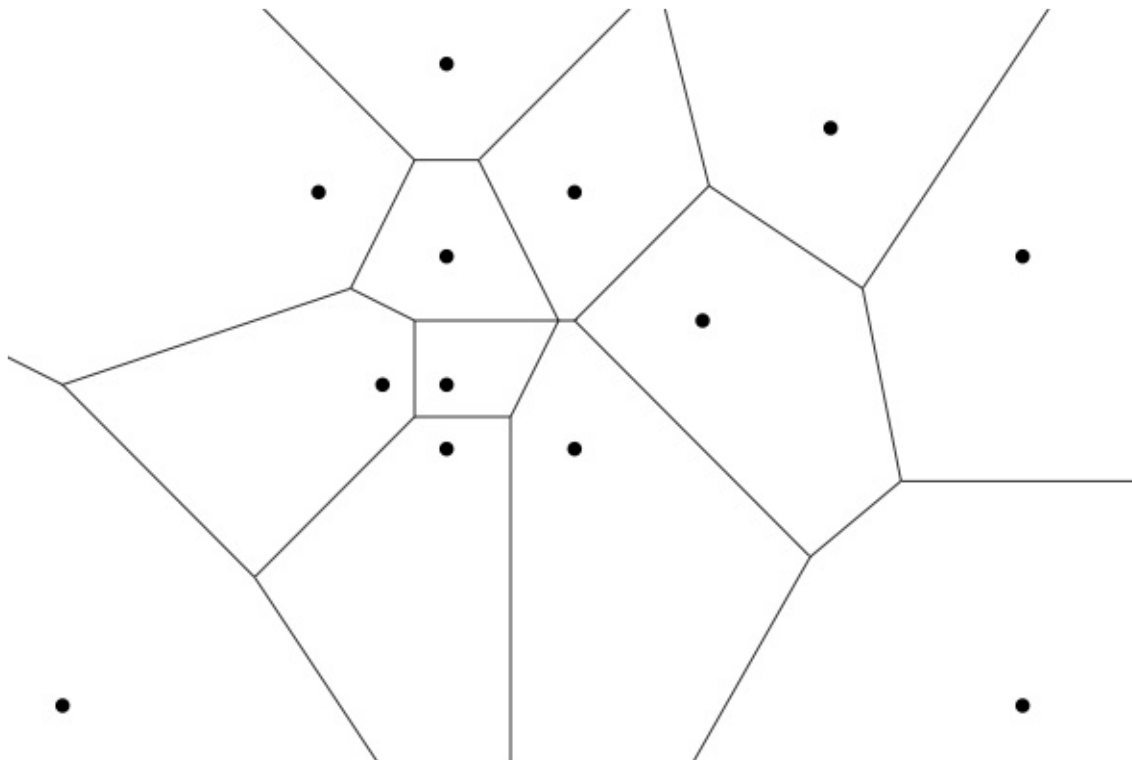
The barebones algorithm for Nearest Neighbor is as follows:

```
Given a new example to label, find the "most similar" example (i.e. the nearest neighbor) in the dataset.  
Predict using the label of the nearest neighbor.
```

How do we compute the distance between examples? What is the distance measure?

For real-valued attributes, we can use the **Euclidean Distance**.

The feature space gets divided into polygonal cells based on the distances to existing examples. The result resembles a **Voronoi Diagram**:



Issues with NN

- One issue with the NN algorithm is that we have to compute the distance to each and every example in the dataset. This can be **time consuming**.
- NN is a **lazy learner**: there is no training. The only processing that happens is when a test example is to be labeled.

- **The scale of attributes:** Attributes may have different scales. This leads to attributes with smaller scales being ignored when we compute the Euclidean Distance. Therefore, it is important to scale the attributes i.e. to bring them to the same scale.
- **The Curse of Dimensionality:** As the number of attributes increases, the amount of training data needed for effective classification (i.e. fast classification) increases exponentially. This impacts the NN algorithm because the examples move further and further apart as the number of dimensions (attributes) increases. Therefore, we need a lot more examples to be able to use NN effectively.
- **High Variance:** Predictions are very sensitive to which examples are in the training set. This also makes it very **sensitive to outliers**. To overcome this, we use the k-NN algorithm. This uses the majority label of the k closest examples to predict the label of a test example. **Note:** The kNN algorithm can also be used for **Regression**! The average (or maybe the weighted average) of the outcomes of the k closest examples can be used to predict the outcome of a test example.

Evaluating a Classifier

Train-Test Split

Given a dataset, we first split it into a **training set** (say 70%) and a **test set** (say 30%). We must make sure to randomly shuffle the examples before splitting, to avoid cases such as all the examples of a certain class being together in the set.

We train using the training set and compute the error (and accuracy) on the test set.

Note: The main aim is to generalize to unseen examples. If we train using the entire dataset, the classifier will most probably **overfit** i.e. it will classify training examples fairly well but wouldn't be able to generalize to unseen test examples.

A good practice is to compute both the **training error** and the **test error**.

If training error \ll test error, the model is probably overfitting!

k-fold Cross-Validation

This is mainly useful when we have a **small dataset**. Splitting a small dataset would reduce the amount of data available for both training and testing.

Instead of splitting the dataset into a train and a test set, do the following:

```
divide the dataset into k equal subsets.  
  
for i = 1 to k:  
    train on examples in all subsets except S_i  
    get hypothesis h_i  
    use h_i to predict labels of examples in S_i  
    compute the error for h_i  
  
compute the average error of all the h_i's
```

(k=5 is common, though k=10, 12 are also used)

This way, every example in the dataset is eventually used for both training and testing.

Note: A version of cross validation can also be used to choose the values of the parameters of a learning algorithm. For example, choosing k for kNN:

- establish a range of possible values for k: say 1, 3, 5, 7, 9
- put some examples from the training set into the validation set
- for each k, train on the examples not in the validation set
- test on the examples in the validation set
- choose the k with the lowest error on the validation set

Leave-one-out cross validation uses $k=N$. It is also called N-fold CV. It is effective, but time consuming.

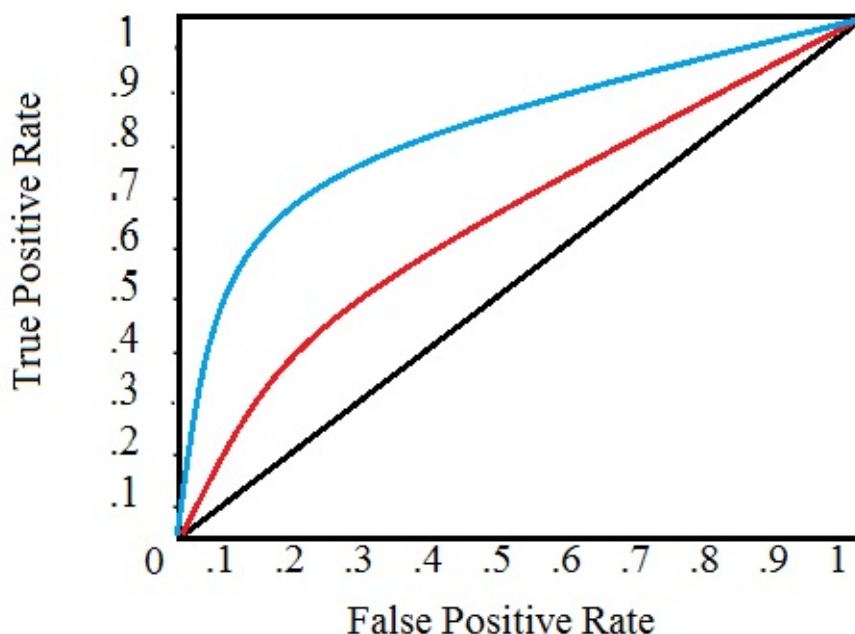
ROC Curve

A confusion matrix gives the TP, FP, TN, FN. We can compute the TPR and FPR.

$$TPR = \frac{TP}{TP+FN}, FPR = \frac{FP}{FP+TN}$$

An ROC curve plots the TPR vs. the FPR at different thresholds.

(for different thresholds applied on $P(+|x)$, the FP, FN, TP, TN change, causing TPR, FPR to change).



The dotted line corresponds to 50% accuracy i.e. as good as random guessing for a binary classification.

The AUC (Area Under Curve) corresponds to the accuracy, so the higher the curve is above the dotted line, the higher is the accuracy of the classifier.

At $TPR=FPR=0$, all examples were labeled negative. At $FPR=TPR=1$, all examples were labeled positive.

At $FPR=0$, $TPR=1$, all examples were classified correctly! (**ROC Heaven!**)

At $FPR=1$, $TPR=0$, all examples were misclassified! (**ROC Hell!**)

At ROC Heaven, $AUC=1$ (max. possible AUC). At ROC Hell, $AUC=0$ (min. possible AUC).

Parametric Estimation

This section discusses how to estimate the parameters of a distribution i.e. μ, σ^2 of the line

$$f(x) = w_0 + w_1x.$$

We denote the parameters by $\Theta = (\mu, \sigma^2)$

The **likelihood** of Θ given a sample X is given by:

$$l(\Theta|X) = \sum_t p(x^t|\Theta)$$

Therefore, the **Log Likelihood** of Θ given a sample X is denoted by:

$$L(\Theta|X) := \log l(\Theta|X) = \sum_t \log p(x^t|\Theta)$$

This assumes that the observations in X are independent.

The **Maximum Likelihood Estimator** (MLE) is given by:

$$\Theta^* := \operatorname{argmax}_{\Theta} L(\Theta|X)$$

Estimating the Parameter P_h of a Bernoulli Distribution

X is a Bernoulli Random Variable.

$$P_h = P[X = 1]$$

For example, consider the following:

Let 1 denote Heads, and 0 denote Tails.

Say $X = \{1, 1, 0\}$

We need to determine Θ i.e. P_h .

We have $l(P_h|X) = P(X|P_h) = P_h * P_h * (1 - P_h)$

More generally, for $X = \{x^t\}_{t=1}^N$, we have:

$$p(X|p_h) = \prod_{t=1}^N p_h^{x^t} (1 - p_h)^{(1-x^t)}$$

It can be proved that the MLE is given by $p_h = \frac{\sum x^t}{N}$.

Estimating the Parameters of a Multinomial Distribution

Consider a die with 6 faces numbered from 1 to 6.

If X is a Multinomial Random Variable, there are k>2 possible values of X (here, 6).

Say X={5, 4, 6}. We can imagine indicator vectors for each observation as [0 0 0 0 1 0], [0 0 0 1 0 0] and [0 0 0 0 0 1].

Say X={4,6,4,2,3,3}. The MLE of x_i i.e. side i shows up, can be given by:

$$p_i = \frac{\sum_{t=1}^N x_i^t}{N}.$$

Estimating the Parameters of a Gaussian Distribution

The MLE for the mean m is $\frac{\sum_t x^t}{N}$ and the MLE for the variance σ^2 is $\frac{\sum_t (x^t - m)^2}{N}$.

However, if we divide by N-1 instead of N (for variance), it is called the **unbiased estimate**.

Bayesian Approach to Parameter Estimation

Treat Θ as a random variable with prior $p(\Theta)$.

According to Bayes' Rule, $p(\Theta|X) = \frac{p(\Theta)p(X|\Theta)}{p(X)}$

- The ML estimate is given by: $\Theta_{ML} = \operatorname{argmax}_{\Theta} p(X|\Theta)$
- The MAP estimate is given by: $\Theta_{MAP} = \operatorname{argmax}_{\Theta} p(X|\Theta)p(\Theta)$
- The **Bayes Estimate** is given by: $\Theta_{BAYES'} = E[\Theta|X] = \int_{\Theta} \Theta p(\Theta|X) d\Theta$ (the integral becomes a summation for discrete values)

Example with a Discrete Prior on Θ

Consider a parameterized distribution uniform on $[0, \Theta]$.

Say the discrete prior on Θ is given by:

$$P(\Theta = 1) = 2/3$$

$$P(\Theta = 2) = 1/3$$

Suppose $X=\{0.5,1.3,0.7\}$

Given X , we know that $P(\Theta|X) = 0$ and therefore, $P(\Theta = 2|X) = 1$. So, the ML, MAP and BAYES' hypotheses are all 2.

Now, suppose $X=\{0.5,0.7,0.1\}$

$$p(X|\Theta = 1) = 1^3 = 1$$

$$p(X|\Theta = 2) = (1/2)^3 = 1/8$$

$$\text{So, } p(X) = P(\Theta = 1)p(X|\Theta = 1) + P(\Theta = 2)p(X|\Theta = 2) = 51/72$$

$$\text{Therefore, } P(\Theta = 1|X) = \frac{p(X|\Theta=1)P(\Theta=1)}{p(X)} = 48/51 \text{ and } P(\Theta = 2|X) = 3/51$$

In this case, the MAP hypothesis is 1 and the ML hypothesis is 1.

The Bayes' hypothesis can be computed as

$$E[\Theta|X] = 1 * (48/51) + 2 * (3/51) = 54/51 = 1.06$$

The **posterior density of x given X** is given by:

$$\begin{aligned} p(x = 0.82|X) &= p(\Theta = 1|X)p(x = 0.82|\Theta = 1) + p(\Theta = 2|X)p(x = 0.82|\Theta = 2) \\ &= (48/51) * 1 + (3/51) * (1/2) = 99/102 \end{aligned}$$

Example with a Continuous Prior on Θ

Assume the data X is drawn from a Gaussian with a known variance σ^2 and an *unknown* mean μ (this is now the Θ).

Assume a Gaussian prior on Θ i.e. $\Theta \sim N(\mu_0, \sigma_0^2)$ and μ_0, σ_0^2 are known.

Then, generate X from $N(\Theta, \sigma^2)$ (this Θ is the mean of the Gaussian from which X was chosen. It is what we need to estimate.)

Given X , we have:

$$\Theta_{ML} = m \text{ (i.e. the sample mean)} = \frac{\sum_i x^i}{N}$$

$$\Theta_{MAP} = \frac{N/\sigma^2}{N/\sigma^2 + 1/\sigma_0^2} m + \frac{1/\sigma_0^2}{N/\sigma^2 + 1/\sigma_0^2} \mu_0$$

$$\Theta_{BAYES'} = \Theta_{MAP}!$$

As $N \rightarrow \infty$, m dominates the weighted sum of m and μ_0 .

Estimates of Mean and Variance of a Distribution (not just Gaussian)

The ML estimate for the mean is m i.e. the sample mean.

The ML estimate of variance is $\frac{\sum_i (x^i - m)^2}{N}$ (this is biased since $E[\sigma^2] < \sigma^2$).

Note that the estimate for variance is **lower** than the actual value because we use the sample mean m to compute it instead of using the actual mean.

However, $\frac{\sum_i (x^i - m)^2}{N-1}$ is an unbiased estimate.

Parametric Estimation for Simple Linear Regression

$$r = f(X) + \epsilon; \epsilon \sim N(\mu, \sigma^2)$$

is a line that minimizes squared error.

$$g(x^t|w_0, w_1) = w_1 x^t + w_0$$

is the line defined by parameters w_0, w_1 . We need to find the line that minimizes squared error, and to do so, we need to compute the values for w_0, w_1 that minimize the squared error.

We have $X = \{x^t, r^t\}_{t=1}^N$

We need to compute $\operatorname{argmin}_{w_0, w_1} \sum_t (r^t - (w_1 x^t + w_0))^2$

To solve for w_0, w_1 , take partial derivatives w.r.t w_0 and w_1 and equate them to 0. We will get 2 equations:

$$\sum_t r^t = Nw_0 + w_1 \sum_t x^t$$

$$\sum_t r^t x^t = w_0 \sum_t x^t + w_1 \sum_t (x^t)^2$$

To solve for w_0, w_1 , we use the closed form solution:

$$W = A^{-1}y$$

$$\text{where } W = \begin{bmatrix} w_0 \\ w_1 \end{bmatrix}, A = \begin{bmatrix} N & \sum_t x^t \\ \sum_t x^t & \sum_t (x^t)^2 \end{bmatrix}, Y = \begin{bmatrix} \sum_t r^t \\ \sum_t r^t x^t \end{bmatrix}$$

Parametric Estimation for Multivariate Linear Regression

$$x = \begin{bmatrix} x_1 \\ \cdot \\ \cdot \\ \cdot \\ x_d \end{bmatrix}$$

We need to find the parameters $W = \begin{bmatrix} w_0 \\ \cdot \\ \cdot \\ \cdot \\ w_d \end{bmatrix}$

so that the linear function $g(x|w_d, w_{d-1}, \dots, w_1, w_0) = w_d x_d + w_{d-1} x_{d-1} + \dots + w_1 x_1 + w_0$

minimizes the square error on the dataset $\{x^t, r^t\}_{t=1}^N$ where $x^t = \begin{bmatrix} x_1^t \\ x_2^t \\ \cdot \\ \cdot \\ x_d^t \end{bmatrix}$

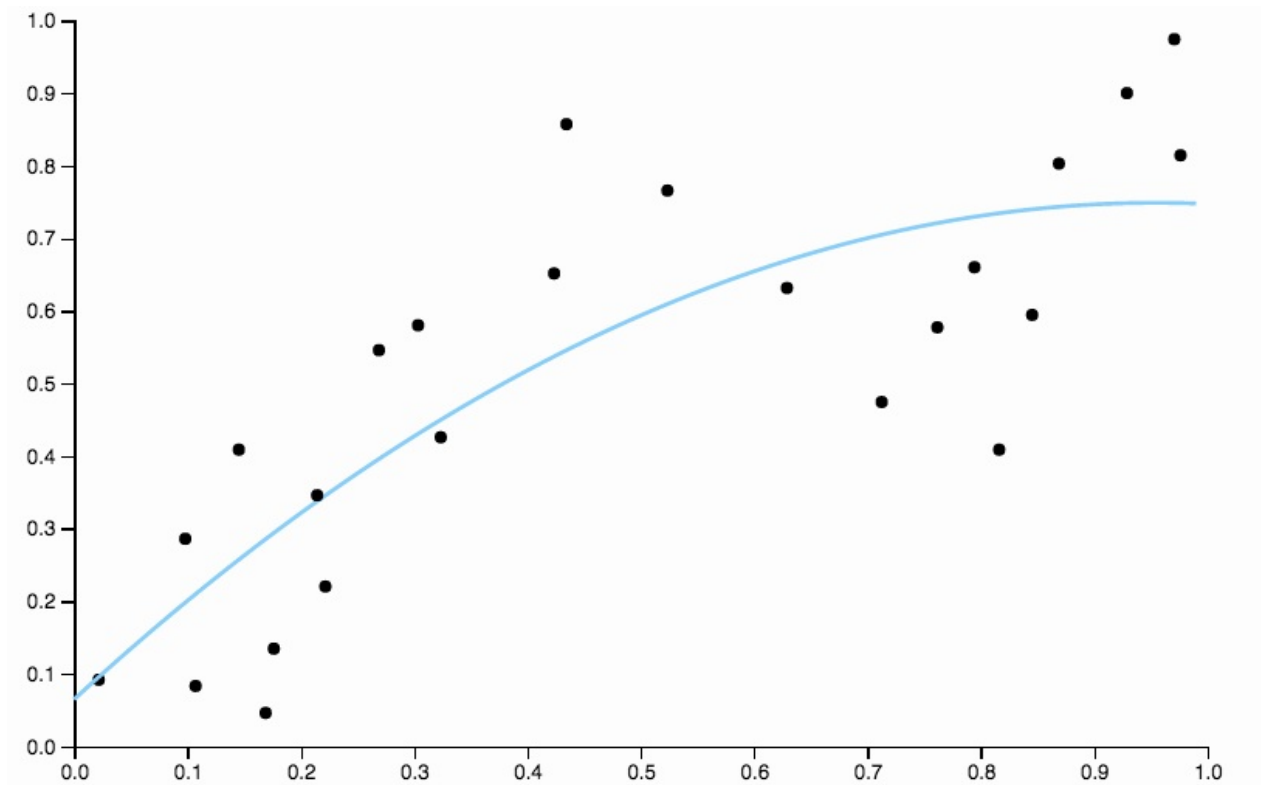
Let $D = \begin{bmatrix} 1 & x_1^1 & x_2^1 & \dots & x_d^1 \\ 1 & x_1^2 & x_2^2 & \dots & x_d^2 \\ \cdot & \cdot & \cdot & \dots & \cdot \\ \cdot & \cdot & \cdot & \dots & \cdot \\ \cdot & \cdot & \cdot & \dots & \cdot \\ 1 & x_1^N & x_2^N & \dots & x_d^N \end{bmatrix}_{N \times (d+1)}$ and $r = \begin{bmatrix} r_1 \\ r_2 \\ \cdot \\ \cdot \\ r_N \end{bmatrix}_{N \times 1}$

Then, $W = (D^T D)^{-1} D^T r$

Sometimes, the inverse doesn't exist. This usually happens when the number of dimensions is too less.

Parametric Estimation for Simple Polynomial Regression

In cases where the data cannot be fit using a linear decision boundary, we may want to use polynomial regression.



Say we want to use a degree 2 polynomial. The equation can be given by:

$$g(x|w_2, w_1, w_0) = w_2x^2 + w_1x + w_0$$

Our aim is to find values for w_0, w_1, w_2 that minimize the squared error $\sum_t (r^t - g(x^t))^2$

Note: Given a dataset $\{x^t, r^t\}_{t=1}^N$ where $x^t \in R$ i.e. where $x^t = [x_1^t]$ (1 dimension), to find the polynomial of degree 2

$g(x|w_2, w_1, w_0) = w_2x^2 + w_1x + w_0$ that minimizes the squared error, we can construct a related dataset with inputs in R^2 (2 dimensions) with the second dimension $x_2^t = (x_1^t)^2$, and then use simple linear regression on this new dataset to obtain w_2, w_1, w_0 that minimize the squared error, and finally output $g(x|w_2, w_1, w_0) = w_2x^2 + w_1x + w_0$ with these w_2, w_1, w_0 values as the best 2 degree polynomial that fits the original dataset.

This can be extended to higher degree polynomials as well.

Parametric Estimation for Multivariate Polynomial Regression

Say we have a dataset with 2 features and it cannot be fit using Linear Regression.

We may want to use Multivariate Polynomial Regression. The equation is given by:

$$g(x_2, x_1 | w_5, w_4, w_3, w_2, w_1, w_0) = w_5 x_2^2 + w_4 x_1^2 + w_3 x_1 x_2 + w_2 x_2 + w_1 x_1 + w_0$$

This is a degree 2 polynomial and we need to estimate $w_5, w_4, w_3, w_2, w_1, w_0$ that minimize the squared error.

Note: In some cases, this can be simplified to a Linear Regression (as shown in Simple Polynomial Regression) by simply adding appropriate features to the dataset.

Bias and Variance of an Estimator

Consider the following estimators of the mean of a distribution. X is an i.i.d. sample from the distribution.

1. $m_1 = \frac{\sum_t x^t}{N}$ (this is the MLE)

2. $m_2 = \frac{x^1 + x^N}{2}$

3. $m_3 = 5$

Now, draw a sample of size N (say $N=3$): $X=\{6,1,5\}$

$$m_1 = 4, m_2 = 11/5, m_3 = 5$$

If we consider the means to be random variables, each of them will have a variance.

Say we want to estimate Θ (here, $\Theta = \mu$ of the distribution from which we are drawing X)

The desirable property of the estimator d of Θ is that the expected value of d must be equal to the quantity we want to estimate i.e. $E[d] = \Theta$. d is then called the **unbiased estimator**.

The **bias of an estimator** ' d ' is given by:

$$b_{\Theta}(d) = E[d] - \Theta$$

If $b_{\Theta}(d) = 0$, d is an **unbiased estimator**.

Is m_2 an unbiased estimator of μ ?

$$m_2 = \frac{x^1 + x^N}{2}$$

$$E[m_2] = E\left[\frac{x^1 + x^N}{2}\right] = \frac{1}{2}E[x^1 + x^N] = \frac{1}{2}E[x^1] + \frac{1}{2}E[x^N]$$

$$\text{Since } E[x^t] = \mu \text{ (by definition), } E[m_2] = \frac{1}{2}\mu + \frac{1}{2}\mu = \mu$$

Therefore, m_2 is an unbiased estimator of the mean μ .

Is m_3 an unbiased estimator of μ ?

$$E[m_3] = E[5] = 5$$

Clearly, $E[m_3] - \mu = 0$ iff $\mu = 5$. Therefore, m_3 is **not** an unbiased estimator of the mean μ .

The **variance of an estimator 'd'** is given by

$$E[(d - E[d])^2]$$

More data leads to lower variance.

m_3 has the least variance (it is always 5!). m_1 has a lower variance than m_2 .

The **square error** of an estimator is given by: $E[(d - \Theta)^2] = (E[d] - \Theta)^2 + E[(d - E[d])^2] =$

$$Bias^2 + Variance$$

Bias and Variance of a Regression Algorithm

Here, we discuss the Bias and Variance of a Regression Algorithm that outputs g .

The expected error of fixed g (w.r.t. random noise ϵ) at x is given by:

$$E[(r - g(x))^2 | x] \text{ (at fixed } x\text{)}$$

$$\text{where } r = f(x) + \epsilon$$

By linearity of expectation i.e. $E[X + Y] = E[X] + E[Y]$, we can show that:

$$E[(r - g(x))^2 | x] = \underbrace{E[(r - E[r|x])^2 | x]}_{(noise)} + \underbrace{(E[r|x] - g(x))^2}_{(squared\ error)}$$

Expected error of g built from random sample X (using our regression algorithm) is given by:

$$E_X = [E[(r - g(x))^2 | x]] = \underbrace{(E[r|x] - E_X[g(x)])^2}_{(bias)} + \underbrace{E_X[(g(x) - E_X[g(x)])^2]}_{(variance)}$$

Bias: Expected error of g on fixed x , that is computed using our algorithm, when we run it on a random sample X .

Variance: The amount g varies based on the sample X .

Bias/Variance Dilemma

As discussed, **bias** is a measure of the expected error (w.r.t. the random sample X on which the algorithm is run) the algorithm will make when it predicts the value r for an input x , and **variance** is a measure of how much the prediction on x will vary depending on which random sample X is used.

An ideal algorithm has **low bias** and **low variance**. However, this is usually not achievable.

Typically, as we *increase the complexity* of g , for example, using Polynomial Regression of degree 2 instead of using Linear Regression, the **bias decreases** (better fit the data) but the **variance increases** (fit varies more with data). This can be related to the concept of **overfitting** since a more complex classifier will better fit the data, but the fit will vary more with the data.

Model Selection

This refers to selecting an appropriate model for the task at hand.

For example, for regression, should we use Linear Regression? Polynomial Regression of degree 2? Polynomial Regression of degree 3? and so on.

Cross-Validation

In very low dimensions, we may be able to visualize/plot the data. We may be tempted to compute the squared errors for Linear Regression and Polynomial Regression and choose the one that performs better. However, **this should not be done!** This is because the squared error may be low on the training data, but may turn out to be extremely high on the test data.

Instead, we must perform **cross-validation**:

- divide the dataset into training and validation sets
- train each model on the training set
- compute the error of the resulting g on the validation set
- choose the model that minimizes the error on the validation set

Regularization

If the number of input variables is large (i.e. large dimension d), then Linear Regression learns a lot of coefficients. Sometimes, these coefficients can be absurdly large or small. This is a sign of overfitting.

In such cases, we can set some coefficients to 0, to simplify g .

We must find the hypothesis (i.e. the linear function) that minimizes the **regularized error function** given by:

$$E^1 = \text{error on data} + \lambda * (\text{model complexity})$$

where 'error on data' can be squared error, λ is a tunable parameter called the **regularization parameter** and the model complexity (for a linear function) can be given by

$$\sum_{i=1}^d |w_i|.$$

The value of λ can be a default value or can be determined using cross-validation.

It can be shown that the hypothesis that maximizes E^1 is the MAP hypothesis, for a suitable prior.

Linear Discriminant Analysis

This is mainly used for classification problems.

Think of it as computing a 'score' for an example which is a weighted sum of the attributes.

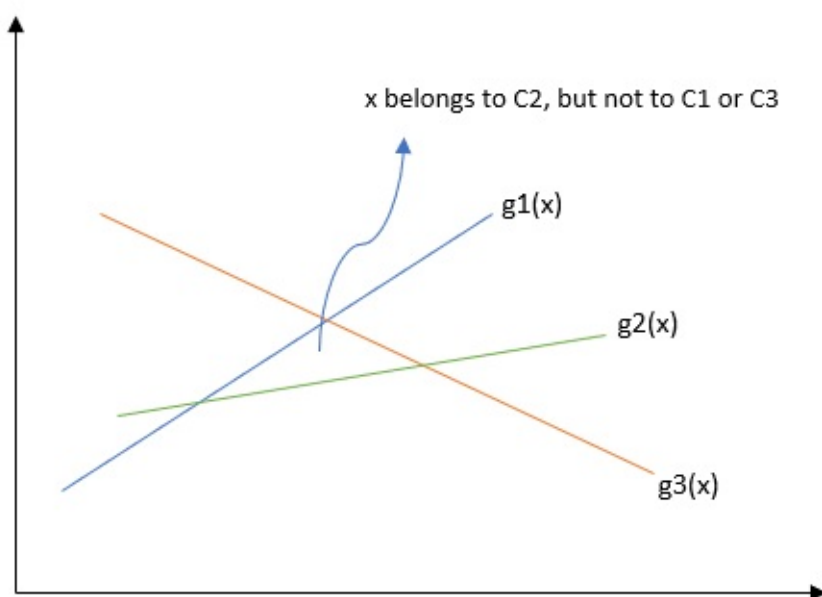
Say we have 3 classes C_1, C_2, C_3 .

The score for class i is given by $g_i(x|w_i, w_{i0}) = w_{i2}x_2 + w_{i1}x_1 + w_{i0}$ where $x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$.

It can be computed using $g_i(x) = w_i^T x + w_{i0}$

Given $g_1(x), g_2(x), g_3(x)$, we must predict the class for x based on the class that maximizes $g_i(x)$ i.e. the $\operatorname{argmax}_i g_i(x)$.

We must learn a $g_i(x)$ that can hopefully make accurate predictions on new examples. This **linear discriminant function** linearly separates the examples that belong to class i and don't belong to class i . (say examples above the line belong to the class and examples below the line do not).



Consider a problem with two classes $C_1(+)$ and $C_2(-)$.

In a **generative approach**, we attempt to learn/model distributions $p(x|+)$ and $p(x|-)$.

In a **discriminative approach**, we don't learn/model $p(x|+)$ and $p(x|-)$. We only attempt to discriminate between + and -.

Let $y \equiv P(C_1|x)$, $1 - y = P(C_2|x)$.

Choose class C_1 if $y > 0.5$, $\frac{y}{1-y} > 1$, or $\log(\frac{y}{1-y}) > 0$. Otherwise, choose C_2 .

The **Logit** or **Log Odds** function is given by $\text{logit}(y) = \log(\frac{y}{1-y})$ for $0 < y < 1$

Its inverse is the logistic function, also called the **sigmoid** function i.e. $\text{sigmoid}(z) = \frac{1}{1+e^{-z}}$

Logistic Regression

This is a **classification** model and is used for real values attributes only, like all other linear discriminant methods.

Consider a problem with 2 classes.

We assume that the log likelihood ratio (log odds) is linear.

$$\log \frac{p(x|C_1)}{p(x|C_2)} = w^T x + w_0^0$$

This assumption only holds if the distributions for C_1 and C_2 are Gaussian, with a shared covariance matrix.¹

We want to classify x into $C_1(+)$ or $C_2(-)$.

Put in class C_1 if $\frac{p(x|C_1)}{p(x|C_2)} > 1$ i.e. if $\log \frac{p(x|C_1)}{p(x|C_2)} > 0$.

Using the property that the inverse of logit is sigmoid, we can show that

$$P(C_1|x) = \frac{1}{1+e^{-(w^T x + w_0)}}.$$

This is the posterior probability that x belongs to C_1 .

Proof:

$$\begin{aligned} \text{logit}(P(C_1|x)) &= \log \frac{P(C_1|x)}{1-P(C_1|x)} = \log \frac{P(C_1|x)}{P(C_2|x)} = \log \frac{P(C_1)p(x|C_1)}{P(C_2)p(x|C_2)} = \log \frac{p(x|C_1)}{p(x|C_2)} + \log \frac{P(C_1)}{P(C_2)} = w^T x + w \\ &= w^T x + w_0 \end{aligned}$$

$$\text{where } w_0 = w_0^0 + \log \frac{P(C_1)}{P(C_2)}$$

Since sigmoid is the inverse of logit, we get:

$$P(C_1|x) = \frac{1}{1+e^{-(w^T x + w_0)}}$$

We predict + if $P(C_1|x) > 0.5$ and predict - otherwise.

Note that when $(w^T x + w_0) \rightarrow +\infty$, $P(C_1|x) \rightarrow 1$ and when

$(w^T x + w_0) \rightarrow -\infty$, $P(C_1|x) \rightarrow 0$

So we can predict + if $w^T x + w_0 > 0$ and - otherwise.

Learning the Weights

We need to learn weights to be able to compute $P(C_1|x) = \frac{1}{1+e^{-(w^T x + w_0)}}$

Consider the data below:

x_1	x_2	r
3	21	1(i.e.+)
6	5	1(i.e.+)
2	9	0(i.e.-)

We need to compute w_2, w_1, w_0 .

Suppose y^t is the real probability that example t is in class C_1 , then the label of example t can be viewed as a random variable.

Assuming that the labelings of the examples are independent, the probability that example 1 is labeled + and example 2 is labeled + and example 3 is labeled - can be given by

$$P(C_1|x^1) * P(C_1|x^2) * (1 - P(C_1|x^3)) = y^1 * y^2 * (1 - y^3)$$

We want to choose w, w_0 (w is a vector that contains w_2, w_1) such that the estimates y^t "match" the labels in the dataset as well as possible, i.e. the must maximize:

$\Pi_{t=1}^N (\hat{P}(C_1|x^t))^{r^t} * (1 - \hat{P}(C_1|x^t))^{(1-r^t)}$ where $\hat{P}(C_1|x^t)$ is the **estimate** of the probability that x^t is labeled 1.

We can simplify the above by using logs. Let $y^t = \frac{1}{1+e^{-(w^T x + w_0)}}$

We need to find $\operatorname{argmax}_{w, w_0} \Pi_t (y^t)^{r^t} (1 - y^t)^{(1-r^t)}$

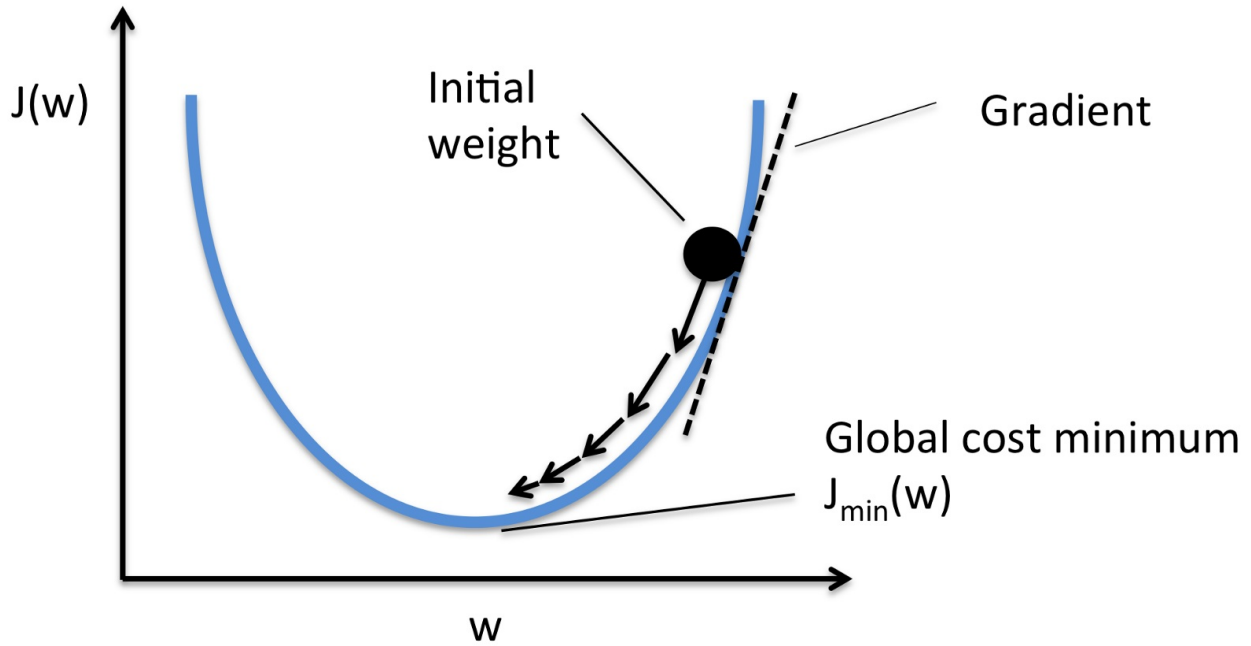
$$= \operatorname{argmax}_{w, w_0} \sum_t [r^t \log y^t + (1 - r^t) \log (1 - y^t)]$$

$$= \operatorname{argmin}_{w, w_0} - \sum_t [r^t \log y^t + (1 - r^t) \log (1 - y^t)]$$

Let us define $Err(w, w_0|X) = -\sum_t [r^t \log y^t + (1 - r^t) \log (1 - y^t)]$, so in other words, we need to find the weights w, w_0 that minimize this error.

To do so, we use an iterative technique known as **Gradient Descent**.

Gradient Descent



We start with **random** weights. Then, we compute the derivative of the error function w.r.t. the weights. Then, we take a step in the direction of steepest descent i.e. in the negative direction of the gradient (since the gradient points to the direction of steepest ascent). This is done iteratively until we can no longer move in a direction that further minimizes the error.

Note that the error function mentioned above has a single minimum (i.e. no local minima, just one global minimum).

For $j \neq 0$,

$$\frac{\partial E}{\partial j} = -\sum_t \left(\frac{r^t}{y^t} - \frac{(1-r^t)}{1-y^t} \right) y^t (1-y^t) x_j^t, \text{ where } y^t = \frac{1}{1+e^{-(w_2 x_2^t + w_1 x_1^t + w_0)}} \text{ and using } \frac{d}{da} y = y(1-y) \text{ for } y = \frac{1}{1+e^{-a}}$$

Therefore, we get:

$$\frac{\partial E}{\partial j} = -\sum_t (r^t - y^t) x_j^t$$

The iterative **update rule** for w_j ($j \neq 0$) is given by:

$w_j \leftarrow w_j + \eta \left(-\frac{\partial E}{\partial w_j} \right)$ where η is the **learning rate**. This can be re-written as:

$$w_j \leftarrow w_j + \eta \sum_t (r^t - y^t) x_j^t$$

Similarly, $w_0 \leftarrow w_0 + \eta \sum_t (r^t - y^t)$

Put generally, $w_j \leftarrow w_j + \eta \Delta w_j$

Pseudocode for Gradient Descent for Logistic Regression

```
# assume x_0 = 1

for j = 0...d:
    w_j = rand(-0.01, 0.01)
    repeat:
        for j = 0...d
            delta_w_j = 0
        for j = 1...N
            theta = 0
            for j = 0...d
                theta = theta + w_j x_j^t # theta = wTx^t + w_0 x_0 where x_0=1
            y = sigmoid(theta)
            delta_w_j = delta_w_j + (r^t - y) x_j^t
        for j = 0...d
            w_j = w_j + n * delta_w_j
    until convergence
```

¹. If x is 1-D, distribution for C_1 is $N(\mu_1, \sigma^2)$, and the distribution for C_2 is $N(\mu_2, \sigma^2)$.

Note that they have **different means**, but the **same variance**. Then, we have

$$\log \frac{p(x|C_1)}{p(x|C_2)} = \log \frac{\frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{(x-\mu_1)^2}{2\sigma^2}}}{\frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{(x-\mu_2)^2}{2\sigma^2}}} = \frac{-(x-\mu_1)^2}{2\sigma^2} + \frac{(x-\mu_2)^2}{2\sigma^2} = \left(\frac{\mu_1 - \mu_2}{\sigma^2} \right) x + \frac{\mu_2^2 - \mu_1^2}{2\sigma^2} \text{ which is linear in } x.$$



Decision Trees

This classifier is commonly used when the data is not linearly separable.

Decision Trees can be used for either just categorical attributes or just numerical attributes or for a mix of both.

Consider the dataset below:

x_1	x_2	label
2.3	4.7	+
1.9	2.8	+
6.1	4.3	-

Each node of the Decision Tree for the above dataset will be of the form $x_i > t$ (for $t \in R$), and will have a binary split (N on the left, Y on the right). The leaf nodes will be labels.



The goal is to generate a **small tree** with **pure** (all examples for that node have the same label) or **mostly pure** (majority examples for that node have the same label) leaves.

Larger trees are more prone to **overfitting**.

It is an NP-hard problem to be able to find the *smallest* tree either in terms of depth or number of nodes or both. So, we use a greedy heuristic. Therefore, there is no guarantee that we are generating the *best* tree.

Some advantages of using Decision Trees:

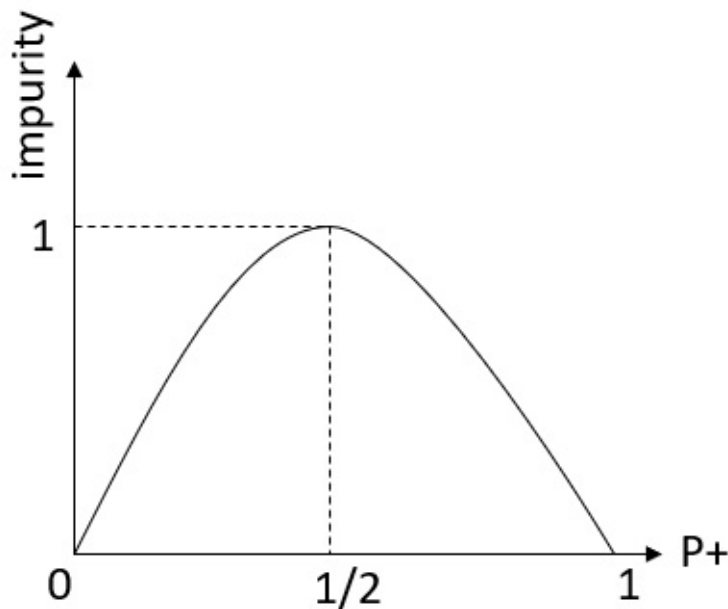
1. Easy to generate trees
2. Human-interpretable i.e. a human can visualize the tree and understand the decisions being made at each node to verify if something is wrong

How to determine the root node?

To determine which attribute to use at the root node, we must compute the impurity for each attribute, and choose the one that is most pure.

Consider labels + and -. Highest purity is when $P_+ = 1$ or $P_+ = 0$ and highest impurity is when $P_+ = 0.5$

where P_+ is the fraction of examples with +.



We use **Entropy** to determine purity.

$$Entropy = -P_+ \log_2(P_+) - P_- \log_2(P_-)$$

If we pick a random example from X , what is the expected entropy of the leaf node where it ends up?

It can be computed as

$$P(\text{split decision}) * Entropy(\text{split decision}) + P(\sim \text{split decision}) * Entropy(\sim \text{split decision})$$

For each possible decision leading to a different partition of X , into X_Y and X_N , choose the decision that minimizes:

$$P[Y] * Entropy(X_Y) + P[N] * Entropy(X_N)$$

where $P[Y] = \frac{|X_Y|}{|X|}$ and $P[N] = \frac{|X_N|}{|X|}$,

and $X_Y = \{x \in X | \text{decision on } x \text{ is } Y\}$, $X_N = \{x \in X | \text{decision on } x \text{ is } N\}$

Note: If we have categorical attributes, instead of Y and N , we will have the values for that attribute (one pre branch).

Preventing Overfitting

There are a few methods to prevent overfitting in Decision Trees:

1. **Stop Growing Tree (Pre-Pruning)**: stop growing the tree before reaching purity in each leaf. Form leaf if either:
 - number of training examples reaching leaf is $< t_1$ (we choose t_1)
 - Entropy of set of training examples reaching node is $< t_2$ (we choose t_2)
2. **Post-Pruning**: grow the tree to purity and then cut off subtrees and replace with leaves. This technique needs a validation set to determine where and how to prune.

Gini Index: an alternative to Entropy

It can be computed using $4P_+(1 - P_+)$

Sometimes, the 4 is omitted.

Using Decision Trees for Regression

In most cases where we use a Decision Tree for regression, the leaf nodes will now have means instead of labels. In some (rare) cases, we can also end up with linear equations in the leaf nodes, similar to the ones we obtain by linear regression.

Bias and Variance

In general, decision trees have low bias and high variance.

To reduce variance, perform **bagging**:

- Take multiple versions of the training set (random subsets)
- Build a separate tree using each subset
- Given a new example to predict, compute the prediction from each tree. Then for classification, predict the majority class. For regression, compute the average of the predictions.

To get a random subset of approx. $2/3$ of the total examples (N), sample N times with replacement. Then, eliminate duplicates and the remaining elements form the random subset.

(Also, at each node, consider only a random subsets of attributes to be used for the decision).

This model is known as a **Random Forest**.

Dimensionality Reduction

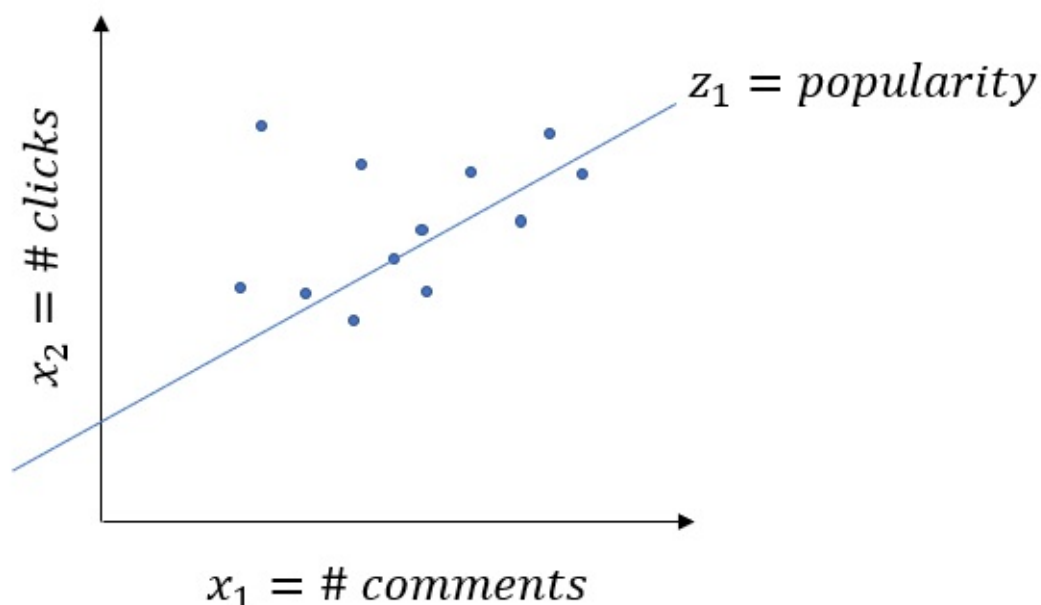
The main aim of dimensionality reduction is to reduce the number of dimensions (attributes/features) that will be used to learn from the data. Learning from too many dimensions can lead to irrelevant model learning. This is the **curse of dimensionality**.

Principal Components Analysis (PCA)

For a given attribute pair, we determine a line that can combine them and then use this new attribute instead of the original attribute pair. Then the next attribute is chosen based on the one that maximizes the variance.

Say we have 3 attributes x_1, x_2, x_3 . We need to find weights that maximize the variance.

- First, center the data around the origin. To do so, subtract the attribute-wise mean of the data points from each example. This will create a new dataset with x'_1, x'_2, \dots, x'_d .
- Then, we choose the line through the origin that maximizes the variance, using a formula (shown below). This line will be a linear combination of the x'_i s. This is z_1 . This preserves the variance in the horizontal direction. For example, consider a YouTube video dataset, if the attributes are #comments and #clicks, then z_1 may model the popularity:



- Now, we choose another line z_2 that is orthogonal to z_1 . This allows us to preserve the variance in the vertical direction.

Therefore, PCA produces attributes that better model the data than the original attributes.

How to find z_1 ?

Compute the sample covariance matrix of the data points $[x'_1, x'_2, \dots, x'_d]$, given by

$\begin{bmatrix} \sigma_1^2 & \text{cov}(x_1, x_2) \\ \text{cov}(x_1, x_2) & \sigma_2^2 \end{bmatrix}$ and compute its eigenvalues and eigenvectors. The highest

eigenvalue is called the **principal eigenvalue** (λ_1).

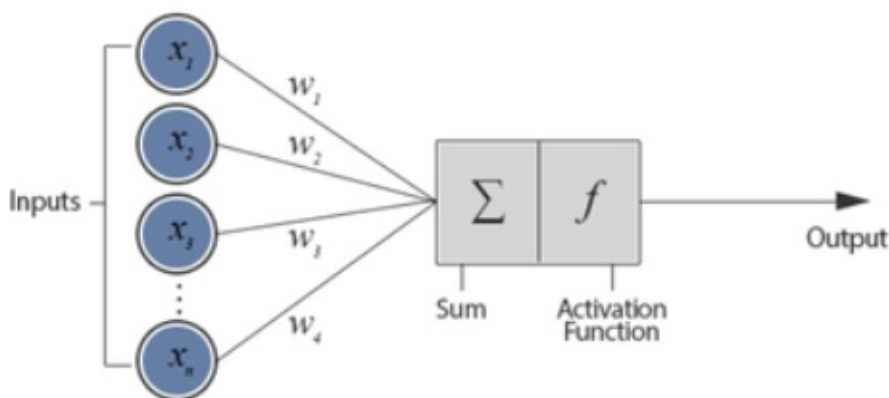
$\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_d$ are the eigenvalues associated with eigenvectors V_1, V_2, \dots, V_d .

Then, use this formula: $z_1 = V_1^T \cdot X'$

Neural Networks

These models aim to mimic the human brain, or are at least inspired by it.


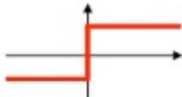





The basic unit in a neural network is a **neuron**. A neuron computes a weighted sum of its inputs and then an activation is computed.



$$X = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ \vdots \\ \vdots \\ x_d \end{bmatrix}, W = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ \vdots \\ \vdots \\ w_d \end{bmatrix} \text{ and } x_0 = 1.$$

The **sigmoid activation function** $\frac{1}{1+e^{-W^T X}}$ gives a probability as an output. The **threshold/step activation function** outputs 1 if $W^T X > 0$ and 0 otherwise. The **linear activation function** simply outputs $W^T X$.

Some common activation functions:

Unit step (Heaviside)	$\phi(z) = \begin{cases} 0, & z < 0, \\ 0.5, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Sign (Signum)	$\phi(z) = \begin{cases} -1, & z < 0, \\ 0, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Linear	$\phi(z) = z$	Adaline, linear regression	
Piece-wise linear	$\phi(z) = \begin{cases} 1, & z \geq \frac{1}{2}, \\ z + \frac{1}{2}, & -\frac{1}{2} < z < \frac{1}{2}, \\ 0, & z \leq -\frac{1}{2}, \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression, Multi-layer NN	
Hyperbolic tangent	$\phi(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	Multi-layer Neural Networks	
Rectifier, ReLU (Rectified Linear Unit)	$\phi(z) = \max(0, z)$	Multi-layer Neural Networks	

A neural network (also known as a **Multi Layer Perceptron (MLP)**) has multiple layers of neurons. The most common problem faced in neural networks is the **credit assignment problem**: it is difficult to determine which neurons are to be given credit/blame for an increase/decrease in accuracy.

Training a Neuron

Batch Training

This is possible when all the data is already available.

Training is done using batches of the data.

Each time the weights are updated, compute gradient for the entire dataset.

```
Repeat until convergence:
  for all i in {1...d}
    w_i = w_i + eta*sum((r^t-y^t)x_i^t)
```

One pass through the dataset constitutes an **epoch**.

On-Line Training

Train using one example at a time. Update the weights only based on that example.

$$w_i \leftarrow w_i + \eta(r^t - y^t)x_i^t$$

This is more efficient than batch training.

Stochastic Training

The idea is to train in an on-line fashion.

```
Repeat until convergence of weights (or some other stopping condition):
  for each x^t (in some order):
    for all i in {1...d}
      w_i = w_i + eta*(r^t-y^t)x_i^t
```


MLP - Multi Layer Perceptron

This is the other name for a neural network.

Advantages

- Good accuracy on even data that is far from linearly separable
- Can learn complicated functions or concepts

Disadvantages

- Danger of overfitting
- Slow to train

Some Notations

Consider a simple 2 layer neural network (the input layer is not counted as a layer):

1 output unit, **H** hidden nodes (+1 dummy bias unit), **d** input nodes (+1 bias unit)

Fully Connected: every node in a layer is connected to every node in the previous layer.

(H+1) + H(d+1) weights to learn.

w_{hj} is the weight on the edge from input node x_j to hidden node h , v_h is the weight on the edge from hidden node h to the output node.

Z_0, Z_1, \dots, Z_H (with $Z_0 = 1$) are the activations from the hidden layer, usually sigmoid i.e.

$$Z_h = \frac{1}{1 + e^{-w^T x}}$$

The **Error Function for Regression** is given by:

$$E(W, v) = \frac{1}{2} \sum_t (r^t - y^t)^2 \text{ i.e. the mean squared error, with } W = \text{weights } w_{hj} \text{ and } v = \text{weights } v_h$$

and $y = v^T Z$ i.e. a linear activation function i.e. $y = v_0 \cdot 1 + v_1 Z_1 + \dots + v_H Z_H$

The **Error Function for Classification** is given by:

$$E(W, v) = - \sum_t r^t \log y^t + (1 - r^t) \log(1 - y^t) \text{ i.e. the cross entropy loss}$$

where $y = \frac{1}{1+e^{-v^T Z}}$ i.e. the *sigmoid function*

Batch Gradient Descent

We must find W, v that minimize the error.

```
1. Initialize  $W$  and  $v$ 
2. Repeat until convergence:
    compute  $v^t$  for each  $x^t$  in training
    update each  $v_h$  and  $w_{h_j}$  by doing:
     $v_h = v_h - \eta * dE/dv_h$ 
     $w_{h_j} = w_{h_j} - \eta * dE/dw_{h_j}$ 
```

We have:

$$\partial E / \partial v_h = \sum_t (r^t - y^t) Z_h^t$$

$$\partial E / \partial w_{h_j} = \sum_t -(r^t - y^t) v_h z_h^t (1 - z_h^t) x_j^t$$

(computed using chain rule i.e. $\partial E / \partial w_{h_j} = \sum_t \partial E / \partial y * \partial y / \partial z_h^t * \partial z_h^t / \partial w_{h_j}$)

This technique is called **backpropagation**.

Regression with Multiple Outputs

The aim here is to predict multiple values, instead of just a single value. This is analogous to multi-label classification where we attempt to predict multiple classes at once.

Reasons to do so:

- less training time
- less number of weights, therefore, less prone to overfitting
- possible relation between values being predicted can be learned

The Error Function is as follows:

$$E = \sum_{i=1}^k \sum_{t=1}^N (r_i^t - y_i^t)^2$$

i.e. the *mean squared error*. (k is the number of values to be predicted)

- can use stochastic gradient descent
- can use mini-batches in the gradient descent

Advice/Tricks and Issues to Train a Neural Network

Adaptive Learning Rate

The idea is to start with a higher learning rate and decrease it as time progresses.

Momentum

$$\Delta w_i^t = -\eta \frac{\partial E^t}{\partial w_i} + \alpha \Delta w_i^{t-1}$$

(t is time). α can be default or computed.

Early Stopping

Too much training could cause overfitting. Stop training when the validation error starts to increase.

Deep Learning

Neural Networks went out of fashion in the 90's when SVMs were introduced.

However, with:

- more data
- more processing power
- better algorithms

neural networks came back, in the form of Deep Learning, i.e. learning with neural networks that have several hidden layers.

The first breakthrough was by a team that used a Deep NN to win an Image Processing competition (ILSVRC).

The main advantage of Deep Learning is that we no longer need to specify the features of the data to the model. The model learns the features automatically. Initial layers learn low-level features while layers further in the network learn higher-level features.

Disadvantages: not intuitive, bad interpretability

Convolutional Neural Networks (CNNs)

These were one of the first Deep NN's to be introduced. They have the following types of layers:

- **Convolutional Layer:** convolves the image with a sliding filter
- **Pooling Layer:** performs dimensionality reduction

CNNs, like standard NN's, are trained using backpropagation.

The **ReLU activation function** is used most often. $\text{ReLU}(x) = \max(x, 0)$.

Droupout Regularization can be used to reduce overfitting.

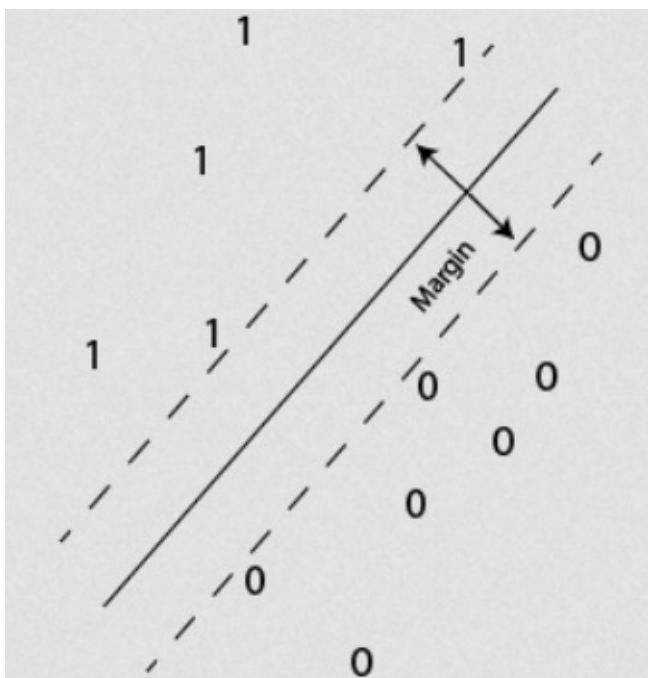
Support Vector Machines (SVMs)

SVMs are also called **kernel methods**.

Given **linearly separable** data (with real-valued attributes), we must find a linear discriminant function that separates the positives and negatives (i.e. the classes) and is as far as possible from any example in the data.

Therefore, SVMs aim to **maximize the distance** of the decision boundary to the closest point in the data set. This distance is called the **margin** of the hyperplane, and is denoted by ρ .

The line that achieves maximum margin is called the **maximum margin hyperplane**.



Computing the Equation for the Maximum Margin Hyperplane

Consider the data below (positive class: +1, negative class: -1)

x_1	x_2	r
1	1	-1
2	2	+1
1	1/2	-1
3	2	+1
2	7/4	+1
3	1	+1

Let $g(x) = w^T x + w_0$

The hyperplane equation is $g(x) = 0$.

The distance from point x to the hyperplane is $\frac{g(x)}{\|w\|_2} = \frac{g(x)}{\sqrt{w_1^2 + w_2^2 + \dots + w_d^2}}$ (note that there is no w_0 in the denominator).

We need to determine w, w_0 such that $w^T x^t + w_0 > 0$ if $r^t = +1$ and $w^T x^t + w_0 \leq 0$ if $r^t = -1$.

For our data, we have the following **linear constraints**:

$$w_1 + w_2 + w_0 \leq 0$$

$$2w_1 + 2w_2 + w_0 > 0$$

$$w_1 + \frac{1}{2}w_2 + w_0 \leq 0, \text{ and so on.}$$

For the maximum margin hyperplane, we have $\frac{g(x)}{\|w\|_2} = \rho$

Suppose we scale the coefficients of g , such that the new $g(x)=1$ (let's call it $\tilde{g}(x)$) for the points closest to the hyperplane. To do so, we must divide the coefficients of g by a certain quantity. Which quantity to use? Maybe the margin ρ ?

Now, replace $g(x)$ by $\tilde{g}(x)$, we have new $g(x) = \pm 1$ for the points closest to the hyperplane. These points are called **support vectors**. Typically, we have at least 3 support vectors, some of which are on the $+$ side and some are on the $-$ side. The intuition behind calling them *support* vectors is that they support the hyperplane in place, and the hyperplane will move if any of them move.

So, for the support vectors, $g(x^t) = +1$ if $r^t = +1$ and $g(x^t) = -1$ if $r^t = -1$. For other examples:

$$g(x^t) > +1 \text{ if } r^t = +1$$

$$g(x^t) < -1 \text{ if } r^t = -1$$

Maximum margin hyperplane on support vectors x satisfies the **canonical form of the maximum margin hyperplane** i.e. $g(x) = \pm 1$. So, $\rho = \frac{\pm 1}{\|w\|_2}$.

Therefore, we must maximize $\frac{1}{\sqrt{w_1^2 + w_2^2 + \dots + w_d^2}}$ subject to the constraints:

$$w^T x + w_0 \geq +1 \text{ for all } x^t \text{ where } r^t = +1$$

$$w^T x + w_0 \leq -1 \text{ for all } x^t \text{ where } r^t = -1$$

The solution gives the weights of the maximum margin hyperplane.

(to simplify, we minimize $w_1^2 + w_2^2 + \dots + w_d^2$).

This is, however, the theoretical computation. SVM softwares compute slightly differently.

SVM softwares often don't directly output the weights. Instead, they output a **dual representation of the hyperplane** $g(x)=0$ where $g(x)=w^T x + w_0$:

Let I be the indices of the support vectors. The SVM software outputs $g(x)$ as a function of the support vectors:

$$g(x) = [\sum_{t \in I} (v_t [x^t])^T . x] + v_0$$

where $x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ i.e. the example to be classified.

Suppose the support vectors are $\begin{bmatrix} 2 \\ 7/4 \end{bmatrix}, \begin{bmatrix} 3 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ say x^{19}, x^5, x^{72} respectively, it will output

coefficients for each support vector i.e. $v_{19} = 3, v_5 = 9, v_{72} = -4$ respectively, and

$$v_0 = 3(\text{say}).$$

The software will, therefore, output $g(x)$. The coefficients of x_1, x_2 in $g(x)$ give the weights w_1, w_2 and the constant term will be w_0 .

So, we have two representations of the maximum margin hyperplane:

The **Primal Representation** $g(x) = \sum_{i=1}^d w_i x_i + w_0$

The **Dual Representation** $g(x) = [\sum_{t \in I} (v_t [x^t])^T . x] + v_0$

($v_t = 0$ for all t where x^t is not a support vector).

SVM with Non-Linearly-Separable Data

The above works when the data is linearly-separable. What if the data is not linearly separable?

Say the data follows $x_1^2 + x_2^2 = 1$. We can make the data linearly separable using

$$z_1 = x_1^2, z_2 = x_2^2.$$

$$g(x) = -x_1^2 - x_2^2 + 1 \Rightarrow g(z) = -z_1 - z_2 + 1$$

Then, the above technique can be used, since the data (in terms of z) is linearly separable.

However, this cannot always be done.

The idea is to **map the data to a new space** where it becomes linearly separable. This new space is an infinite-dimensional space.

Let $z^t = \phi(x^t)$ where ϕ is a mapping function.

Then, we have:

$$g(\phi(x)) = [\sum_{t \in I} v_t \phi(x^t)] \cdot \phi(x) + v_0 = [\sum_{t \in I} v_t [\phi(x^t) \cdot \phi(x)]] + v_0$$

Suppose we have an easy way to compute $K(x, y) = \phi(x) \cdot \phi(y)$, for x, y in the original space, K is called a **kernel function**. Then, we have:

$$g(\phi(x)) = [\sum_{t \in I} v_t [K(x^t, x)]] + v_0 \text{ ----- (1)}$$

The **kernel trick** is to avoid computing ϕ in the new space, by using a kernel function.

Some commonly-used kernel functions:

- **Linear Kernel:** $K(x, y) = x \cdot y$
- **Polynomial Kernel** (of degree d): For $d=2$, $x_1, x_2 \rightarrow x_1^2, x_2^2, x_1 x_2, x_1, x_2$ Consider

$$K(x, y) = K(x_1, x_2, y_1, y_2) \text{ Let } \phi^{new}(x) = \begin{bmatrix} 1 \\ \sqrt{2}x_1 \\ \sqrt{2}x_2 \\ \sqrt{2}x_1 x_2 \\ x_1^2 \\ x_2^2 \end{bmatrix} \text{ (this makes it easier to compute),}$$

then, $K^{new}(x, y) = \phi^{new}(x) \cdot \phi^{new}(y) = (1 + x \cdot y)^2$ In general, if the degree is d , the kernel

function is $K(x, y) = (1 + x \cdot y)^d$ The hope is that the data becomes linearly separable by a d -degree polynomial in the attributes. Another version without lower order terms is

$K(x, y) = (x \cdot y)^d$ d is a tunable parameter.

- **Gaussian Kernel (Radial Basis Function RBF Kernel):** $K(x, y) = e^{-\frac{\|x-y\|^2}{2\sigma^2}}$ where

$\|x - y\|^2$ is the L2 norm i.e. basically the distance between x and y . σ is a tunable parameter. Using a Gaussian kernel is analogous to computing a distance-weighted sum (similar to KNN). It is the most suitable kernel when the data forms a checker-board pattern. To use a Polynomial kernel for the same data, we would need to use high degree d . The Gaussian kernel is the most commonly-used kernel. It can also be

computed as $K(x, y) = e^{-\gamma\|x-y\|^2}$ where $\gamma = \frac{1}{2\sigma^2}$. Small σ means a narrow Gaussian, while small γ means a flat Gaussian.

In general, we predict + if $g(\phi(x)) \geq 0$ and predict - otherwise. (from (1)).

Note: If the number of support vectors returned by the SVM software is not much smaller than the size of the training set, it is a clear sign of overfitting.

Soft-Margin Hyperplanes

The previously discussed hyperplane refers to a **hard-margin hyperplane**. A soft-margin hyperplane allows examples to be in the margin region, as well as on the wrong side of the hyperplane. To accommodate for the examples that are on the wrong side of the hyperplane, we penalize by the distance of those examples.

For a hard-margin hyperplane, we minimize $\|w\|^2$ subject to:

$$w^T x^t \geq 1 \text{ if } y^t = +1, \text{ and}$$

$$w^T x^t \leq -1 \text{ if } y^t = -1$$

Equivalently, $y^t * (w^T x^t) \geq 1$ for all $x^t \in X$

For a soft-margin hyperplane, we do the following:

$$\min \|w\|^2 + \lambda \sum_t \epsilon^t, \text{ subject to:}$$

$$y^t * (w^T x^t) \geq 1 - \epsilon^t; (\epsilon^t \geq 0 \text{ for all } t)$$

λ is the penalty parameter. Note that a large penalty may cause overfitting.

Ensemble Learning

This is the concept of combining multiple learners.

Different learners could use:

- different algorithms
- different parameters
- different training sets

There is, however, little theoretical justification for just generating hypotheses using different learning algorithms. Instead, run the same algorithm on different training sets.

Two common ensemble learning techniques:

- bagging
- boosting

Boosting

Use multiple **weak** learners (i.e. the learner's test accuracy is slightly better than random, say 51%).

For example, use several Decision Stumps (i.e. a Decision Tree with just a root and leaves).

- First, assign equal weights to all training examples.
- Run a weak learner on the training set, and get a hypothesis h .
- Now, increment the weights (using a certain formula) of the **misclassified examples** (if an example has weight 3, it is repeated 3 times, so now there will be 3 such examples in the updated dataset).
- Now, train a new weak learner on the updated dataset.
- Keep repeating this, until a good accuracy is achieved.
- **In effect, each new weak learner learns from the mistakes of the previous weak learner.**
- In the end (we decide when to stop), we will have i hypotheses. Compute a weighted vote (using a certain formula) to determine the prediction for an example.

Theoretically, overfitting may be a concern, but it isn't usually a problem if the learners are weak.

Bagging

A weighted vote of **weak** learners is used to compute the prediction.

Unsupervised Learning

In this kind of learning, we do not have labeled training data.

The most common unsupervised problem is **clustering**. There are also other unsupervised problems such as dimensionality reduction.

Some applications of clustering include:

- color quantization (16 million colors to 256 colors)
- directed marketing (different ads for different groups of people)
- pre-processing stage in supervised learning (in certain cases where it is clear that the examples are coming from well-defined groups)

K-Means Clustering

- Assume that the examples are d -dimensional vectors
- **Goal:** group the examples into k clusters
- Cluster i will be defined by a point in R^d called \mathbf{m} (this is also called the **cluster representative**)
- Assign each example to the cluster with the closest representative using Euclidean distance (if two cluster representatives are equally close, choose the lower valued one)
- Recompute cluster representatives (by computing the mean of all the examples in the cluster i.e. computing the centroid)
- Repeat the previous two steps until convergence i.e. till the representatives stop changing (**it is guaranteed to converge in a finite number of steps because there are a finite number of ways in which n points can be assigned to k clusters i.e. k^n ways**)

Intuitively, a clustering is good if all the points are near their cluster representatives. The aim is to minimize the intra-cluster distance and to maximize the inter-cluster distance.

The error function is as follows:

$$\sum_t ||x^t - \rho(x^t)||^2$$

where $\rho(x^t)$ is the cluster representative of the cluster to which x^t belongs.

Initially, we do not know the number of clusters. So, we can start with 2 clusters and keep increasing the number of clusters until the clustering seems satisfactory. Finding the optimal number of clusters is an NP-hard problem. Therefore, the K-Means clustering algorithm (heuristic) is not guaranteed to find the optimal k cluster representatives, but works well in practice.

Note, we can always get 0 error by choosing $k=n$, but this is not our aim. We must manually determine an optimal number of clusters for our data.

The final converged clustering may change based on changing the initial k cluster representatives. So, how do we choose the initial k cluster representatives? Some approaches:

- choose k random points from the dataset
- choose k random points from the domain (they need not be from the dataset)

We could re-run the clustering with different initial representatives, and choose the one with lower error/the one that seems to better satisfy our purposes.

Probabilistic Clustering

Clusters form because of one simple reason: the points are being drawn from different distributions. Points that come from the same distribution tend to get clustered together.

Say we have data for men's heights and women's heights. The data points will clearly be from two Gaussian distributions.

Sometimes, we do not know how many distributions the data points were drawn from. So, we could experiment with different k values.

Say we have two distributions with means μ_1, μ_2 and standard deviations σ_1, σ_2 .

$$\text{Then, } p(X|\mu_1, \mu_2, \sigma_1, \sigma_2) = \prod_{x \in X} \left[\frac{1}{\sqrt{2\pi}\sigma_1} e^{-\frac{(x-\mu_1)^2}{2\sigma_1^2}} + \frac{1}{\sqrt{2\pi}\sigma_2} e^{-\frac{(x-\mu_2)^2}{2\sigma_2^2}} \right]$$

We need to estimate the parameters μ_1, μ_2 for the Mixture of the Gaussians (i.e. a **Gaussian Mixture Model** (GMM)), assuming we have a known σ ($\sigma_1 = \sigma_2 = \sigma$) and a uniform prior.

$$\operatorname{argmax}_{\mu_1, \mu_2} p(X|\mu_1, \mu_2)$$

To do so, we use an approach called **Expectation Maximization** (EM)

EM in the Context of GMMs

Here, we discuss EM in the context of a Mixture of 2 Gaussians, with the assumptions that we have a known σ ($\sigma_1 = \sigma_2 = \sigma$) and a uniform prior.

Assume each data point is actually a triple of the form (x, z_1, z_2) , where x is given but z_1, z_2 are unobservable.

$z_1 = 1$ if the point was generated from G_1 (Gaussian 1), and 0 otherwise. Similarly for z_2 .

Algorithm

1. Pick random initial values for $h = (\mu_1, \mu_2)$
2. **E Step**: for $j=1$ to 2: calculate the expected value $E[z_j]$ for each x , assuming that the current hypothesis is correct

$$E[z_1] = 1 \cdot p(x \text{ was generated from dist. 1}) + 0 \cdot p(x \text{ was generated from dist. 2})$$

$$= p(x \text{ was generated from dist. 1}) = P(G_1|x) \text{ in general,}$$

$$E[z_j] = P(G_j|x) = \frac{p(x|G_j)P(G_j)}{p(x)} \text{ here, } E[z_1] = \frac{p(x|G_1) \cdot \frac{1}{2}}{p(x|G_1) \cdot \frac{1}{2} + p(x|G_2) \cdot \frac{1}{2}} \text{ (since we assume uniform}$$

$$\text{prior, } P(G_1) = P(G_2) = \frac{1}{2}) = \frac{p(x|G_1)}{p(x|G_1) + p(x|G_2)} \text{ and } E[z_2] = \frac{p(x|G_2)}{p(x|G_1) + p(x|G_2)}. \text{ So,}$$

$$E[z_1] + E[z_2] = 1.$$

3. M Step:

for $j= 1$ to 2 :

update the means, using a weighted average:

$$\mu_j = \frac{\sum_{x \in X} x \cdot P(G_j|x)}{\sum_x P(G_j|x)}$$

4. Repeat 2 and 3 until convergence.

EM may not converge to a global optimum, but will at least converge to a local optimum, in a reasonable amount of time.

Note: This is soft-clustering. If we wanted to perform hard clustering, we would make z_1 or z_2 equal to 0 or 1, depending on whichever was bigger than the other, for each example in each iteration.

Why Use K Means instead of EM?

- It does not make any underlying assumptions about the distributions from which the data was drawn. So, if we aren't convinced that our data comes from Gaussians, we may want to steer clear from EM.
- Also, EM gives a soft clustering by default, while KNN gives a hard clustering.

Reinforcement Learning

In this kind of learning, the computer learns to choose the best course of action under a given circumstance, with the aim of maximizing a **reward**. It makes use of continuous feedback to judge the correctness of its decisions.

One issue is that this feedback is not instant: it can be **delayed**.

Credit Assignment is another issue: how to determine which action is to be given credit/blame?

State changes are an issue as well, because the state of the problem may/will change after every action is performed.

Reinforcement Learning is commonly used in learning to play games such as Chess, Checkers, Go etc.

Usually, a **grid world** is used with (s_t, a_t) pairs, i.e. state at time t , and action at time t . A **policy** tells the learner what action to perform in a given state. The best policy is one that maximizes the reward. Therefore, *the aim of Reinforcement Learning is to learn this optimal policy*.

The **Exploration-Exploitation Trade-off**: cost is incurred while exploring for better alternatives. This is common in the **Bandit Problem**.

Note that it isn't necessary to maximize the sum of rewards at each step, but it is important to maximize the final reward.

$Q^*(s, a)$ is the maximum discounted reward that can be obtained if the learner started in state s and did action a as its first step. (* denotes optimal)

Steps (Q-Learning for Deterministic Worlds)

For each (s, a) , initialize $\hat{Q}(s, a) = 0$

Observe current state s

Do forever:

- select an action a and execute it
- receive reward r
- observe new state s'

- update the estimate of $\hat{Q}(s, a)$ as follows: $\hat{Q}(s, a) = r + \gamma \max_{a'} \hat{Q}(s', a')$
(γ is used for discounting the reward, the max term is the maximum discounted reward starting in s' , assuming $\hat{Q} = Q^*$)